UNIVERSITE PARIS VIII
DEPARTEMENT D'INFØRMATIQUE

INFØRMATIQUE ET ARTS GRAPHIQUES

INTRODUCTION A LA PROGRAMMATION PL1600

LE TRAITEMENT DE CARACTERES

Support de Cours

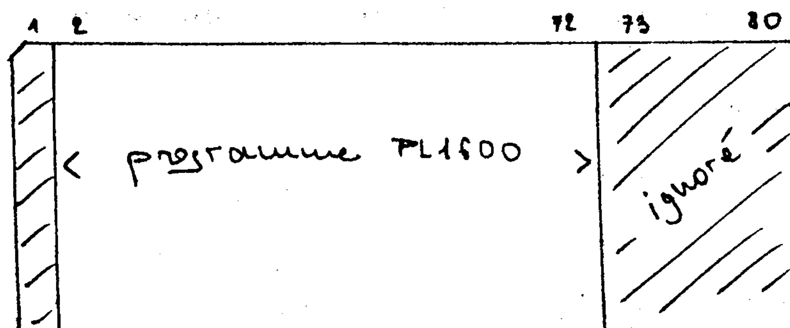
J. CHAILLOUX
MARS 75

Elements du Langage

Un programme PL1600 est composé de mots (Unités Syntaxiques). Ces mots sont lus sur des enregistrements physiques (ex: cartes perforées). Le format est libre. i.e. Il peut y avoir autant d'U.S. par E.P. que l'on veut.

contraintes:
- une U.S. ne peut être coupée sur 2 E.P.
- seuls les 72 1ers caractères de chaque E.P. sont pris en compte.
- le 1er caractère à un statut spécial (validation de compilation).

Dans la pratique, un programme perforé sur carte aura le format suivant:



Une U.S. peut être:
- un symbole de base
- une constante
- un identificateur
- un commentaire

Toute U.S. peut être encadrée d'un nombre quelconque d'espaces.

LES SYMBØLES DE BASE

Ils sont formés d'1 ou plusieurs caractères speciaux :

les séparateurs	␣	(espace)
	,	(virgule)
	;	(point-virgule)
	.	(point)
	:	(deux-points)
les spécificateurs	$	de nombre binaire
	'	de nombre hexadecimal
	"	de chaîne de caractères
	&	d'indirection
	@	d'adresse
	!	de directive au compilateur
	<<	de commentaire
les parenthèses	(	
	)	
les operateurs arithmetiques	+	addition
	−	soustraction
	/	division
	*	multiplication
les operateurs de relation	=	egalité
	/=	non egalité
	>	plus grand
	>=	plus grand ou égal
	<	plus petit
	<=	plus petit ou égal
les operateurs d'affectation	:=	chargement
	:=:	échange

LES CONSTANTES

Elles représentent des valeurs sur lesquelles va travailler le programme.

Les nombres entiers décimaux :

suite de chiffres décimaux (de 0 à 9)

ex : 1 23 32765

Les nombres entiers héxadécimaux :

suite de chiffres héxadecimaux (de 0 à 9, de A à F) précédée du symbole (") (spécificateur de nombre héxadécimal). Un chiffre héxadécimal represente 4 bits.

ex : "12F0 represente en binaire 0001 0010 1111 0000

"3CE7 " " " 0011 1100 1110 0111

Les nombres entiers binaires :

suite de chiffres binaires (0 ou 1) précédée du symbole de base ($) (spécificateur de nombre entier binaire).

ex : $11011 $0001 $111100

Les nombres flottants décimaux

une suite de chiffres decimaux

et/ou { contenant un (.) decimal (pas en 1re position)
se terminant par (E) suivi d'un nombre entier signé ou non.

ex : 124. ⟶ 124,0

0. ⟶ 0,0

12E4 ⟶ $12,0 \times 10^4$

1.12E-5 ⟶ $1,12 \times 10^{-5}$

chaîne de caractères :

suite de caractères quelconques encadrée du symbole de base ("). Si le caractère " doit apparaître dans la chaîne, on le met 2 fois. (seules ces constantes peuvent contenir des espaces).

ex	"ABCZERT"	suite ABCZERT
	"ABnnERT"	suite ABnnERT
	"AB""/*"""	suite AB"/*"
	""""	suite "

constante d'adresse :

nom d'un identificateur précédé du symbole de base @. La valeur d'une constante d'adresse est l'adresse (l'endroit en mémoire) où est implantée cet identificateur.

ex : @IDENT

LES IDENTIFICATEURS

C'est le nom donné par le programmeur à un élément du programme (constante, variable, instruction....).
C'est une suite de caractères alphanumériques
- dont le 1er caractère est toujours une lettre,
- d'au maximum 15 caractères.

Attention : certains identificateurs sont reservés :
les mots-clés (ex: IF DØ END ....)
voir la liste en annexe.
Il est interdit de les employer.

LES CØMMENTAIRES

Ils sont précédés du symbole de base ⓒⓒ, et se terminent à la fin de l'Enregistrement Physique.
(sur carte, ils font passer à la carte suivante).

QUELQUES DEFINITIØNS

Un programme est un ensemble de déclarations et d'instructions.

Les déclarations servent à décrire les éléments sur lesquels va travailler le programme et les associent à un identificateur.

Les instructions décrivent les opérations à effectuer sur ces éléments pour réaliser un traitement (calcul, test, contrôles ....).

Un bloc est une suite de déclarations puis d'instructions (La partie déclaration est facultative). Un bloc sera limité par des délimiteurs de début et de fin de bloc par ex: BEGIN <bloc> END

Une procédure est la déclaration d'un traitement qui peut être :

- global (ou principale) on parlera en PL1600 de MAIN PRØCEDURE
- partiel (sous-programme) on parlera de PRØCEDURE

LES DECLARATIONS

Elles servent à :

- décrire
- donner un nom (identificateur)
- parfois à réserver l'emplacement mémoire,

des données d'un traitement.

La description de ces données portera sur :

- leur nature : constante
  variable
  étiquette ....
- leur mode d'accès : direct (simple)
  indexé (en tableau)
  indirect (en pointeur)
  indirect indexé (en pointeur de tableau)

La notion de déclaration est très liée à celle de bloc car :

- elles apparaissent en tête de bloc (avant les instructions)
- les noms des données (identificateurs) ne sont connus que dans le bloc où ils sont déclarés.

Le même identificateur peut servir pour nommer des données differentes si ces données ne sont pas déclarées dans le même bloc.

Il est obligatoire en PL1600 de déclarer TOUTES les données.

Toute déclaration sera terminée par le symbole de base point-virgule ; .

LES SECTIØNS de DØNNÉES

L'Adressage mémoire sur T1600

Au niveau langage machine T1600, toutes les instructions tiennent sur des mots de 16 bits. Il en résulte que pour les instructions avec "référence mémoire" (qui se servent du contenu d'un mot mémoire) on ne pourra pas coder dans l'instruction l'adresse entière d'un mot mémoire. En effet pour adresser par exemple 32 K mots, il faut déjà 15 bits.

L'adressage sur T1600 est <u>basé</u> :

une adresse c'est l'adresse contenue dans un registre de base
+
un déplacement (±128) par rapport à cette base.

= l'adresse cherchée.

Il y a 3 registres de base.

Il faudra 2 bits pour coder ce registre
+ 8 bits pour coder le déplacement } = 10 bits.

A un instant donné, les 3 registres de base étant chargés, on a accès à 3x256 mots mémoire. On peut avoir accès à toute la mémoire en modifiant les contenus des registres de base.

Au niveau programmation PL1600

① La zone mémoire de 256 mots accessible par un registre de base s'appelle une SECTIØN.

② les 3 registres de base ont pour nom

CØMMØN (ou comme petit nom	RC)
LØCAL	RL
WØRKING	RW

On parlera donc de CØMMØN SECTIØN, de LØCAL SECTIØN ou de WØRKING SECTIØN.

③ Il faut en tête des déclarations d'un bloc indiquer dans quelle section il faudra reserver de la place pour les données

④ Il faut en tête des instructions indiquer les sections que l'on va utiliser

Déclaration de section de donnée

Ouverture

```
    ⎧ COMMON  ⎫
  • ⎨ LOCAL   ⎬  SECTION  <nom>
    ⎩ WORKING ⎭
```

L'ouverture d'une section indique au compilateur que toutes les données suivantes seront rangées dans la nouvelle ~~la~~ section de nom <nom>.

<nom> est un identificateur d'au plus 6 caractères.

Réouverture

```
    ⎧ COMMON  ⎫           continue
  • ⎨ LOCAL   ⎬  SECTION  <nom>
    ⎩ WORKING ⎭
```

La réouverture d'une section indique que toutes les données suivantes seront rangées dans une section dont on avait déjà commencé le chargement.

Les types des sections ouvertes n'ont aucune importance. Habituellement si l'on a qu'une section à ouvrir ce sera une SECTION LOCAL

ex: ~~SECTION LOC~~

```
•LOCAL SECTION LOC
```

la KSTØRE

En plus des 3 registres de base, le T1600 à un registre (le registre K) qui pointe sur une section gérée en PILE. Le type de cette section est KSTØRE (rangement par K).

Cette pile est utilisée pour :

- le système T1600 (zone de manoeuvre pour les entrées/sorties ....)
- stocker les adresses de retour des sous-programmes....

Dans tout programme PL1600, il est obligatoire de réserver une (et une seule) section de ce type.

Réservation de KSTØRE

.KSTØRE SECTIØN <nom> RES <nombre> ;

<nom> est le nom donné à cette section.
<nombre> est le nombre de mots qu'il faut lui réserver (au maximum 256)

ex: .KSTØRE SECTIØN PØLLUX RES 50;

DECLARATIØN de CØNSTANTES

Une constante est une donnée qui reste inchangée tout au long d'un traitement. Il est interdit de modifier une constante.

```
CØNSTANT <nom> = <valeur>
```

<nom> est un identificateur quelconque.

<valeur> est une constante numérique de n'importe quel type.

ex:

```
CØNSTANT  DIX = 10;
CØNSTANT  MASC = 'FF00;
CØNSTANT  ØK = "ØK";
CØNSTANT  VALBIN = $11011;
CØNSTANT  ADBUF = @BUF;
```

DECLARATIØN DE VARIABLE

Une variable est une donnée de programme qu'on pourra modifier, tester ...
Il faut préciser pour chaque variable - son type
- son mode d'accès.

Type

	occupe	représente
BYTE	8bits (1 octect)	1 caractère (code ASCii)
WØRD	16 bits (1 mot)	1 valeur entière
FLØAT	32 bits (2 mots)	1 valeur flottante

mode d'accès direct (variable simple)

<type> <nom de variable> [, <nom de variable> ...] ;

La variable contient 1 élément du type donné. On a accès à cet élément par le nom de la variable.

déclaration: WØRD I, J; BYTE ØCT; accès: I ØCT J

mode d'accès indexé (tableau)

ARRAY n <type> <nom de variable> [, <nom de variable> ...] ;

La variable contient n éléments du type donné. n est une constante entière. On a accès à 1 élément de la variable par son nom suivi d'un indice entre parenthèses. Cet indice est le numero de l'élément que l'on veut sélécter. Attention : le 1er element a pour indice 0.

déclaration: ARRAY 10 WØRD TAB; accès au 1er element: TAB(0)
4ème element: TAB(3)
⋮

mode d'accès indirect (pointeur)

PØINTER <type> <nom de variable> [, <nom de variable> ...] ;

La variable contient l'adresse d'un élément du type donné. On a accès à l'adresse elle-même par le nom de la variable, et à l'élément dont l'adresse est dans cette variable par le nom de la variable précédé du symbole de base &.

déclar: PØINTER WØRD PTW; accès à l'adresse : PTW accès à l'élément : &PTW

mode d'accès indirect post-indexé (pointeur de tableau)

PØINTER ARRAY <type> <nom de variable> [, <nom de variable> ...] ;

La variable contient l'adresse d'un tableau d'éléments du type donné. On a accès à l'adresse elle-même par le nom de la variable et à 1 élément du tableau dont l'adresse est dans cette variable par le nom de la variable suivi d'un indice entre parenthèse ; précédée du symbole de base &.

décl: PØINTER ARRAY BYTE PØCT ; accès à l'adresse : PØCT ; accès à 1 element : &PØCT(3)

Initialisation

On peut initialiser une variable à sa declaration

.... <nom de variable> = (<valeur des elements> / <chaîne de caractères>) ....

Les valeurs sont des constantes séparées par des virgules, factorisables par *n (valeur).

ex:
```
WØRD I=(0), J=(4);
ARRAY 14 BYTE MESER =("ERREUR␣NØ␣:␣00");
ARRAY 10 WØRD T=(0,1,2,*3(4),0,*3(1));
```

DECLARATIØN D'ETIQUETTE

Une étiquette est un nom donné à une instruction. Elle apparait, suivie du symbole de base (:) devant une instruction.

ex: E12 : I:=0 ;
étiquette instruction

Elles servent à faire des branchements (rupture de sequence). La déclaration d'une étiquette se fait exceptionnellement en tête de la partie instruction du bloc où elles apparait.

LABEL <nom d'étiquette> [, <nom d'étiquette> ...] ;

Cette déclaration n'est toutefois necessaire que si l'on utilise cette étiquette avant son apparition (devant une instruction) dans un bloc plus interne à celui où elle apparait (problème des "références avant").

Dans la pratique on déclarera toutes les etiquettes en tête du bloc où elles apparaissent.

ex:

```
PRØCEDURE PRØC
    :                  } partie déclaration.
  LABEL L1, L2 ;
    :
   L1: ....            } partie instruction.
    :
   L2: ....
    :
END;
```

DECLARATIØN DE PRØCEDURE

Une procédure décrit un traitement:

1- Traitement principal (global)

```
MAIN PRØCEDURE <nom de procedure>
                    < bloc >
END.
```

le point . après le END signifie la fin de la compilation.
le nom de la procédure principale ne doit pas dépasser 6 caractères.

2- Traitement partiel (sous-programme)

```
PRØCEDURE <nom de procédure> [ ( <paramètre> [, .....] ) ]
                    <bloc >
END;
```

3- Procédure standard

Une procédure standard est une procédure déjà compilée se trouvant dans la bibliothèque standard. C'est le cas de certaines entrée/sortie, des conversions ...

```
REF PRØCEDURE <nom de procedure> ;
```

ex:

```
REF PRØCEDURE LICART;
REF PRØCEDURE IMPRIM;
```

EXEMPLE DE DECLARATIØNS

```
MAIN PRØCEDURE PP

•KSTØRE SECTIØN PØLLUX RES 50;

•LØCAL SECTIØN LØC
        CØNSTANT MAXLG = 60;
        WØRD PØ = (100);
        ARRAY 80 BYTE BUF;
        ARRAY 10 WØRD TAB = (0, *9(1));
        REF PRØCEDURE IMPRIM;

        PRØCEDURE P1
        •LØCAL SECTIØN CØNTINUE LØC
                WØRD AUX;
              | instructions de P1
        END;        << DE P1

        PRØCEDURE P2 (VAR)
        •LØCAL SECTIØN CØNTINUE LØC
                PØINTER WØRD VAR;
              | instructions de P2
        END;        << DE P2
    | instructions de PP
END.    << DE PP
```

LES INSTRUCTIONS

Elles se terminent par le symbole de base point-virgule (;)
ou par un délimiteur de fin de bloc (END, ELSE)

Il y a des instructions simples et des instructions composées.

Instructions Simples

- L'Affectation (modification de variable)
- branchement (rupture de séquence)
- appel de procédure (rupture vers sous-programme)

Instructions Composées

Elles sont équivalentes à une instruction simple (même si elles ouvrent des blocs).

- Ouverture de bloc	BEGIN <bloc> END;
- Test	IF .... THEN <bloc> [ELSE <bloc>] END;
- Boucle	DØ ...; <bloc> END;
- Aiguillage	CASE .... END;

Toutes ces instructions apparaissent dans la partie instructions d'un bloc, ce bloc étant une procédure ou une autre instruction composée.

C'est l'imbrication des procédures et des instructions composées qui donne à un programme PL1600 sa structure de bloc.

`.USING <registre de base> {IS | =} <nom de section> [, .....] ;`

Les données sont rangées dans des SECTIØN.
L'instruction .USING permet d'indiquer au compilateur les sections sur lesquelles va travailler la partie instruction suivante.

Cette instruction ne peut apparaître qu'au début de la partie instruction d'un bloc.

Elle est obligatoire :
- pour les blocs des procédures
- pour les blocs qui contiennent des déclarations.

2 écritures possibles :

<registre de base> = <nom de section> si on a pas encore indiquer au compilateur d'utiliser cette section.
par ex : le .USING de la MAIN PRØCEDURE.

<registre de base> IS <nom de section> si on a déjà indiquer au compilateur d'utiliser cette section.
par ex : le .USING des PRØCEDURE.

On peut en tête des instructions de n'importe quel bloc repositionner les registres de bases (changer les sections accessibles) par .USING ... = ... ;

AFFECTATION

Une affectation permet de modifier une variable.

1ère forme

<variable 1> [, <variable 2> ....] := <expression>

L'expression à droite du symbole de base := est calculée puis sa valeur est rangée dans toutes les variables de la liste à gauche du := .

Cette liste de variables est une suite de noms de variables (que l'on veut voir "affectées" de la valeur de l'expression), séparés par des virgules.

Les variables peuvent avoir n'importe quel ~~type~~ mode d'accès.

Comment nommer les variables :

- simples (mode direct) : le nom suffit.
- en tableau (mode indexé) : le nom du tableau suivi d'un indice entre parenthèses. L'indice est le numéro de l'élément du tableau que l'on veut désigner.
- pointeurs (mode indirect) : le nom du pointeur précédé du symbole de base &
- pointeurs de tableau (indirect post-indexé) : le nom du pointeur de tableau précédé du symbole de base & et suivi d'un indice entre parenthèse.

Par la suite <constante> désignera
- une constante numérique
- un identificateur déclaré CØNSTANT

<variable> désignera une variable qui peut avoir n'importe quel mode d'accès.

Ecriture d'un Indice

un indice c'est
- <constante>
- <variable>
- <variable> {±} <constante>

Seules ces 3 formes sont acceptées.

Attention : - le 1er element d'un tableau à l'indice 0
- A l'exécution aucun contrôle n'est fait sur la valeur d'un indice.

Ecriture d'une Expression

une expression est composée de
- termes :
  - <constante>
  - <variable>
  - expression entre parenthèses
- d'opérateurs binaires (ex : + - / ....)
- d'opérateurs unaires (ex: NØT SWAP ...)

L'évaluation d'une expression se fait de la gauche vers la droite sans priorité d'opérateur.

Un operateur binaire porte sur
- le résultat de l'expression qui le précède
- le terme suivant.

Un opérateur unaire porte uniquement sur le résultat de l'expression qui le précède.

Opérateurs binaires

+	addition	arithmétiques
-	soustraction	
*	multiplication	
/	division	
AND	intersection	logiques
ØR	union	
XØR	disjonction	

Operateurs unaires

NEG	négation arithmétique
NØT	inversion logique
SWAP	échange des 2 bytes d'un mot.

Beaucoup d'autres opérateurs existent, en effet on a accès en PL1600 à toutes les opérations du calculateur (décalages de toutes sortes, recherche de bit ....)

exemple d'affectation (1ere forme)

```
I, J := 0;

VAL := A + B;

VAL := A * B / C AND 'FF;

C := A + (B * C) SWAP ØR $11011;
```

2eme forme

```
{ INCR }  <variable>
{ DECR }
```

C'est forme d'affectation permet d'ajouter 1 (INCR) ou de retrancher 1 (DECR) à n'importe quelle variable.

ex: equivalent à :

```
INCR I ;                    I := I+1;
DECR TAB (8);               TAB (8) := TAB (8) - 1 ;
```

3eme forme

modification d'un bit d'une variable quelconque.

```
{ SET    }
{ RESET  } BIT <numéro> ØF <variable>
{ NEGATE }
```

<numéro> est le numéro du bit que l'on veut modifier ; il doit être compris entre 0 et 15 .

SET remet le bit désigné à 1.
RESET remet le bit désigné à 0 .
NEGATE inverse le bit désigné.

ex : equivalent à :

```
SET BIT 3 ØF V;                V:= V AND 'EFFF ØR '1000;
RESET BIT 15 ØF TAB (3);       TAB(3) :=TAB(3) AND 'FFFE ;
NEGATE BIT 7 ØF K;
```

— —

test sur bit   ([NØT] BİT <numéro> ØF <variable>)

La variable <variable> est quelconque
<numéro> est compris entre { 0 et 15 pour des variables WØRD.
{ 0 et 31 pour des variables LØNG ou FLØAT.

La condition BİT est vraie si le bit <numéro> de <variable> est à 1.
NØT BİT est vraie si le bit <numéro> de <variable> est à 0.

Condition ØR

Une condition ØR est une suite de conditions simples séparées par l'opérateur ØR; elle est vraie si au moins 1 des conditions simples est vraie.

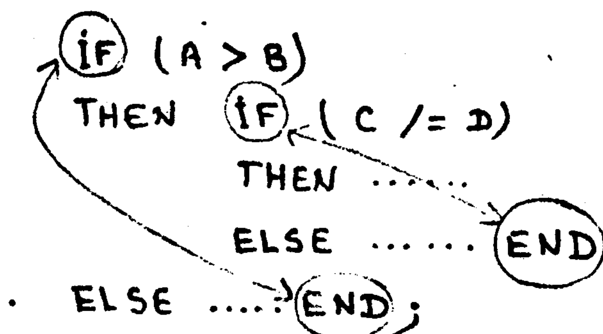
Condition AND

Une condition AND est une suite de conditions simples séparées par l'opérateur AND; elle est vraie si toutes les conditions simples sont vraies.

```
IF (A = B) THEN ...... END;                         Condition simple

IF (A /= B*C + (3*Z)) ØR (Z = 0)                    Condition ØR
   THEN ..... END;

IF (A > B) AND (BIT 3 ØF V) THEN .... END;          Condition AND
```

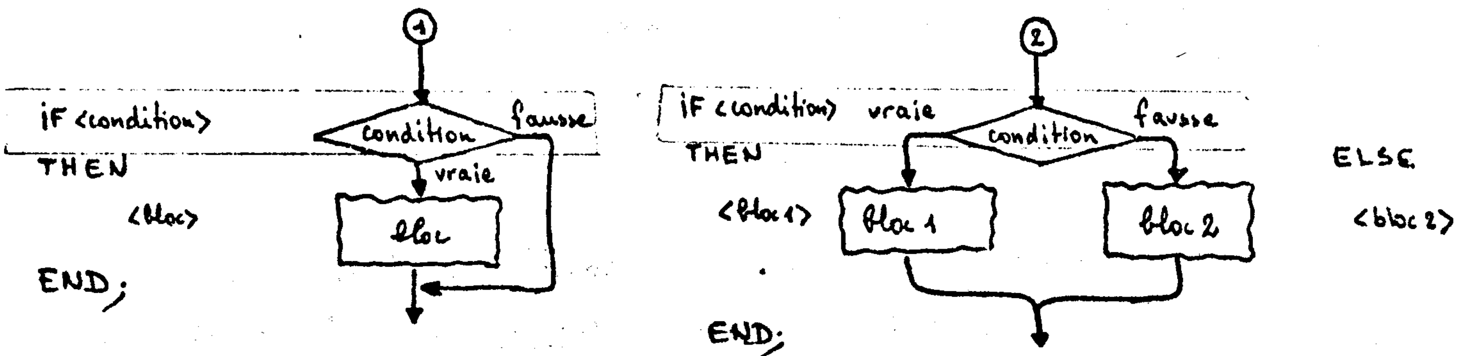


Les instructions IF peuvent s'imbriquer sans précaution particulière, chaque IF étant fermé par un END.

INSTRUCTION COMPOSÉE CONDITIONNELLE

① IF <condition> THEN <bloc> END;
② IF <condition> THEN <bloc1> ELSE <bloc2> END;

Cette instruction permet d'exécuter un bloc défini dans l'instruction sous certaines conditions.



Les blocs peuvent contenir un nombre quelconque d'instruction de n'importe quel type. Il y a 3 types de conditions :

Conditions simples

Elles permettent de tester le contenu d'une variable ou bien 1 bit d'une variable.

test sur variable (<variable> <operateur de relation> <expression>)

la variable <variable> peut être quelconque
les opérateurs de relation sont :

=	égal
/=	non égal (différent)
>	plus grand
>=	plus grand ou égal
<	plus petit
<=	plus petit ou égal

L'expression peut être quelconque mais du même type que la variable.

BRANCHEMENTS - RUPTURE DE SEQUENCE

1ère forme

```
GØTØ <étiquette> [ ØN <condition ØR> ]
```

branchement { inconditionnel / conditionnel (si ØN...) } à <étiquette>.

<condition ØR> s'écrit comme pour l'instruction IF.

2ème forme

```
EXIT [ <étiquette> ] [ ØN <condition ØR> ]
```

branchement { inconditionnel / conditionnel (si ØN...) } sur l'instruction suivant le END de l'instruction composée { en cours (s'il n'y a pas d'étiquette) / nommée par <étiquette> }.

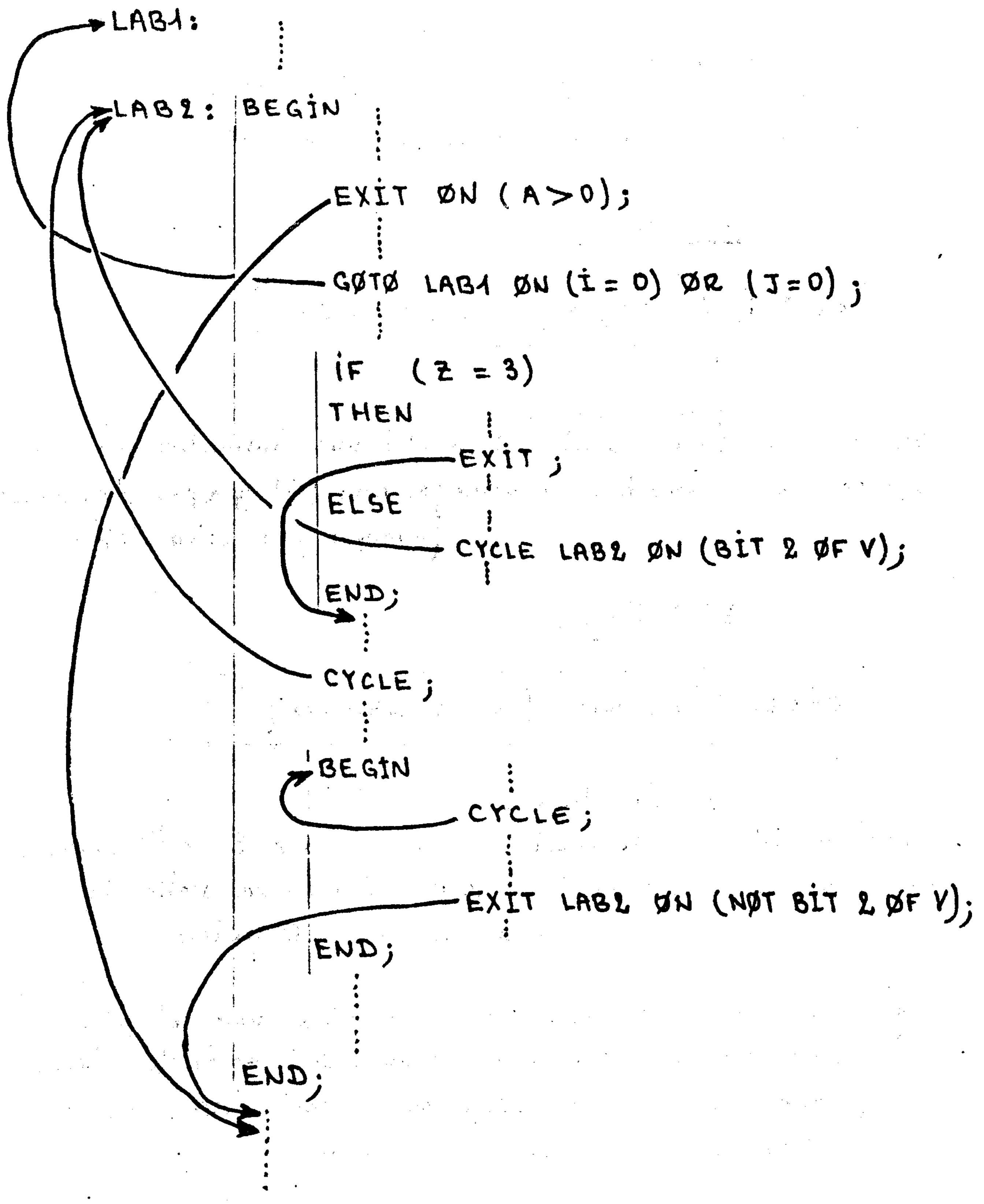
3ème forme

```
CYCLE [ <étiquette> ] [ ØN <condition ØR> ]
```

branchement { inconditionnel / conditionnel (si ØN...) } sur la 1ère instruction de l'instruction composée { en cours (s'il n'y a pas d'étiquette) / nommée par <étiquette> }.

<étiquette> pour EXIT ou CYCLE doit être une étiquette d'instruction composée. On peut ainsi sortir (ou boucler) de plusieurs instructions composées imbriquées mais sans sortir de la procedure en cours.

EXEMPLES DE BRANCHEMENTS



APPEL ET RETOUR DE PROCEDURE

Appel de procédure (sous-programme)

CALL <nom de procédure> [(<par1> [, <par2> ...])]

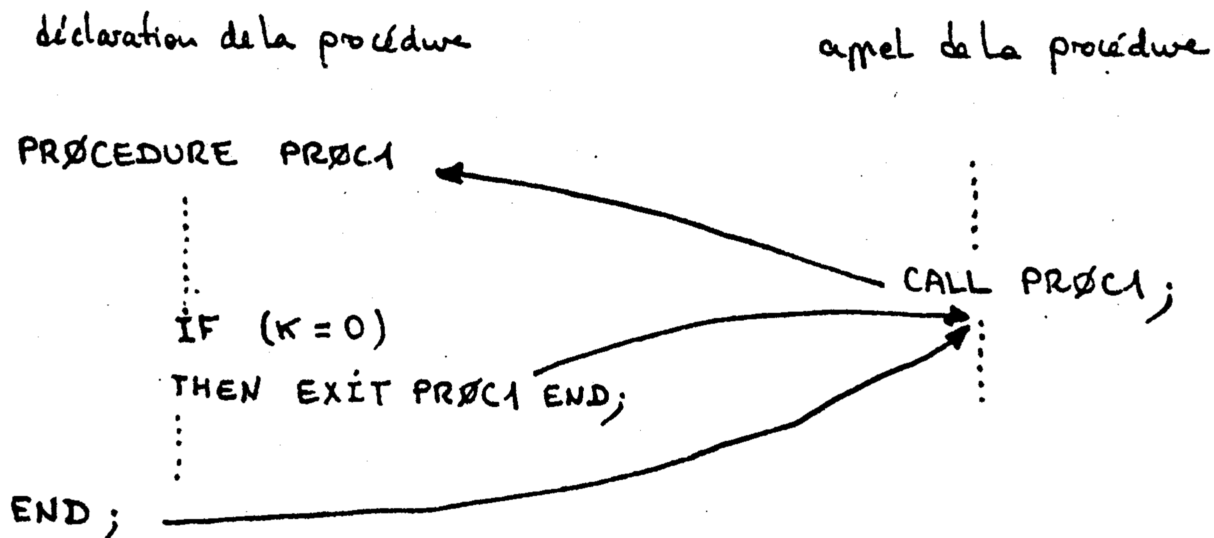
L'appel d'une procédure provoque son éxécution. On peut fournir à l'appel une liste d'arguments séparés par des virgules, entre parenthèse. Ces arguments peuvent être des expressions quelconques.

Retour de procédure (sous-programme)

EXIT <nom de procédure>

La fin d'une procédure et le retour à l'instruction qui suit l'instruction d'appel (CALL) est effectuée :
- après l'éxecution de la dernière instruction de la procédure,
- par l'instruction EXIT <nom de procédure>

On ne peut sortir (par un EXIT) que de la procédure en cours.



INSTRUCTION COMPOSEE { DE CHOIX / D'AIGUILLAGE

```
CASE ( <variable> ) / <nombre>  ØF
                 :
                 :   suite de <nombre>
                 :   instructions
  END;
```

Cette instruction permet de choisir 1 instruction (quelconque) à executer parmis d'autres, en fonction de la valeur d'une variable. L'instruction executée est celle dont le numéro d'ordre est égal à la valeur de la variable. Le numéro d'ordre commence à 1.

<nombre> est le nombre d'instructions possibles (comprises entre le ØF et le END).

si la variable = 0 aucune instruction n'est éxécutée,
si la variable < 0 ou $>$ <nombre> ça fait n'importe quoi.

exemples

```
CASE (K) / 4  ØF
1ère  |  A := B+C;
2ème  |  iF (A < B)
      |     THEN A := B
      |     ELSE B := A  END;
3ème  |  BEGIN  A := 0;
      |         C := 9  END;
4ème  |  CALL  PRØC1;
END;
```

equivalent à:

```
iF (K = 1)
THEN A := B+C
ELSE iF (K=2)
     THEN iF (A<B)
             THEN A := B
             ELSE B := A  END
     ELSE iF (K = 3)
          THEN BEGIN A := 0;
                     C := 9 END
          ELSE iF (K=4)
               THEN CALL PRØC1
               END
          END
     END
END;
```

INSTRUCTION COMPOSÉE { DE BOUCLE / D'ITÉRATION

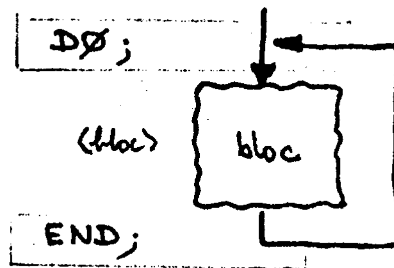
DØ <clause> ; <bloc> END ;

Provoque l'éxécution du bloc un certain nombre de fois. <clause> permet de controler ce nombre (ou la condition de reéxecution du bloc).

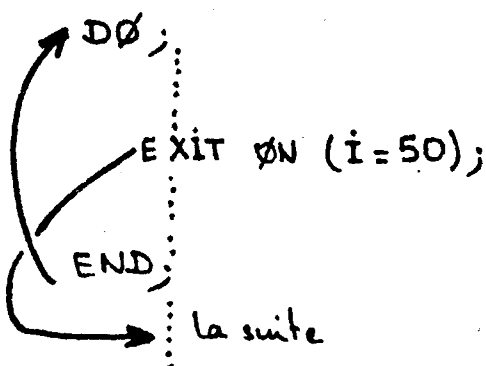
Il y a 4 formes de <clause>

clause vide

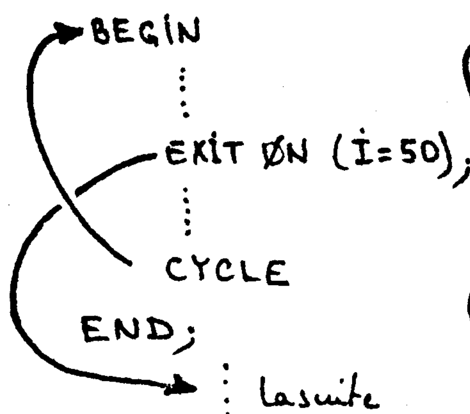
On ne met rien dans <clause>. C'est une boucle illimitée, on reéxécute le bloc indefiniment. On ne peut en sortir que par un GØTØ ou un EXIT.

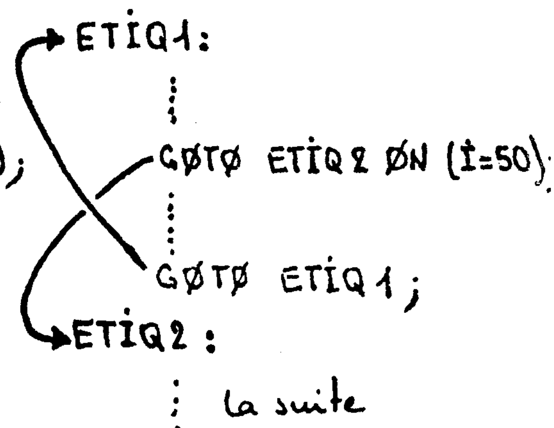


exemple:



équivalent à :



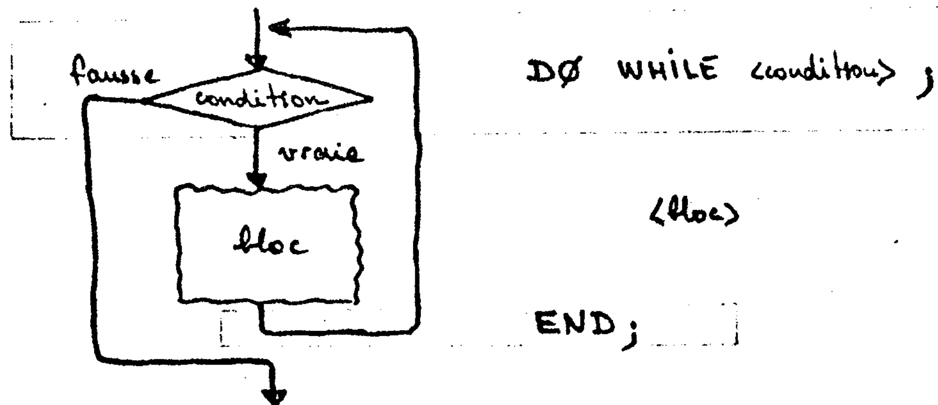


(Cette forme n'est pas tout-à fait equivalente à l'exemple: il n'y a pas d'ouverture de bloc)

clause WHILE (clause tant que)

DØ WHILE <condition> ; <bloc> END;

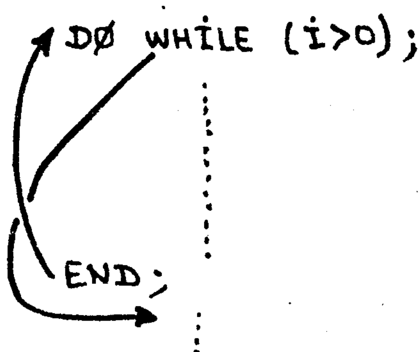
Le bloc est exécuté tant que la condition est vraie.
La condition s'écrit comme pour l'instruction IF (condition simple, ØR ou AND).

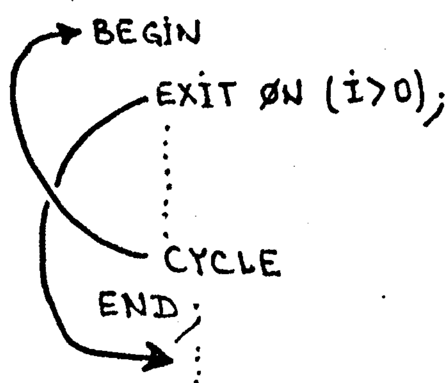


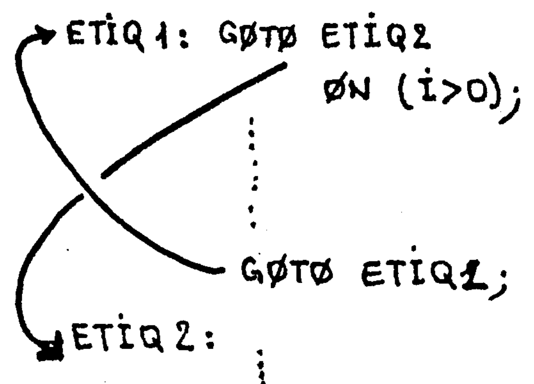
Attention : le test est effectué avant l'exécution du bloc
⇒ si la condition est fausse dès la 1ère boucle, le bloc ne sera jamais exécuté.

exemples

equivalent à





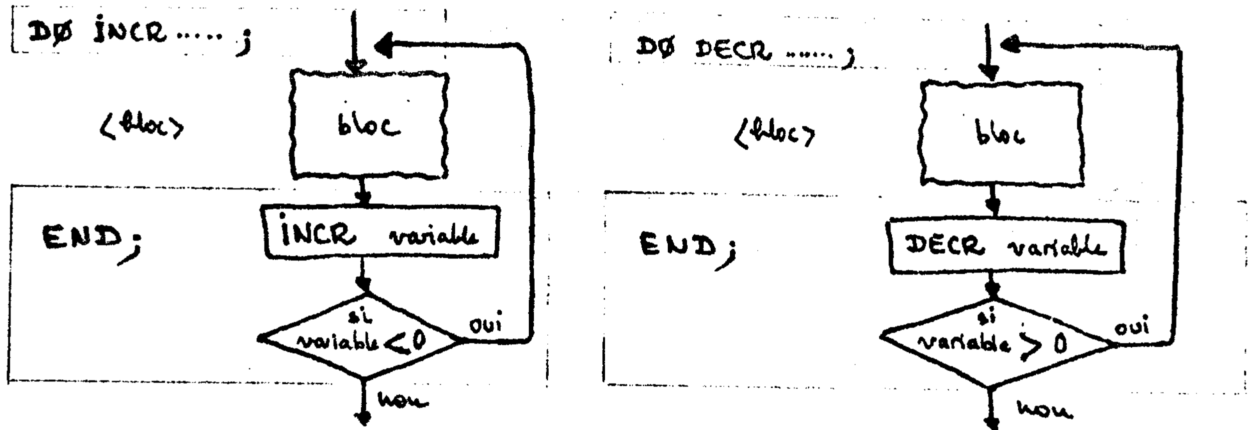


(même remarque qu'à l'exemple précédent)

Clause TIMES (clause nombre de fois)

DØ { INCR / DECR } <variable> TIMES ; <bloc> END;

le bloc va être executé n fois (n est la valeur absolue de <variable>).



Attention: le test se fait <u>après</u> l'execution du bloc ⇒ le bloc est executé au moins une fois.
quand on sort normalement de cette boucle, la variable = 0 .

exemple:

```
I:= -20;
DØ INCR I TIMES;
   :
END;
```

équivalent à :

```
I:= -20;
BEGIN
   :
   INCR I;
   CYCLE ØN (I < 0)
END;
```

Cette boucle est executée 20 fois pour I variant de -20 à 0.

```
I:= 20;
DØ DECR I TIMES;
   :
END;
```

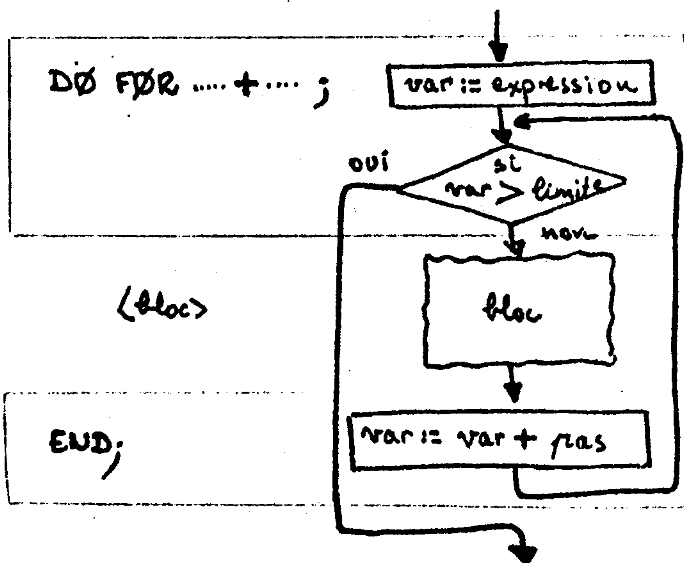
```
I:= 20;
BEGIN
   :
   DECR I;
   CYCLE ØN (I > 0)
END;
```

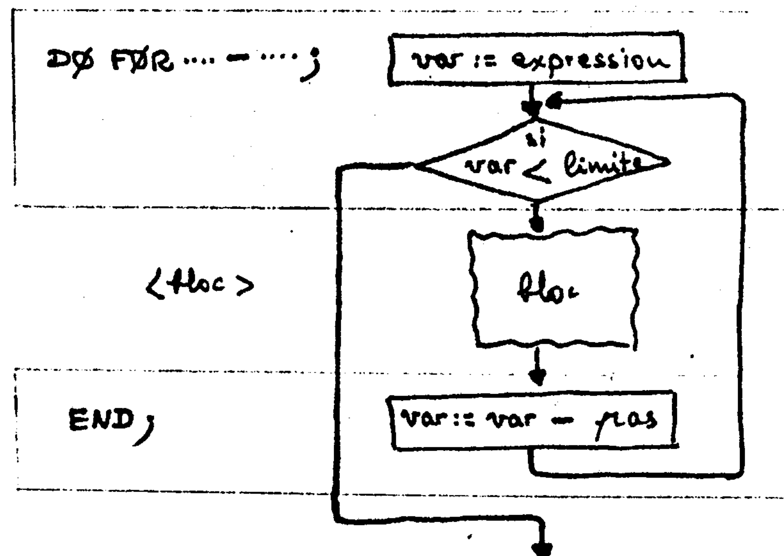
Clause FØR (avec variable de contrôle)

DØ FØR <assignation> STEP {+/−} <pas> UNTIL <limite> ; <bloc> END;

C'est une généralisation de la précédente.

On va
1. donner une valeur de départ à une variable (qui va contrôler la boucle) avec <assignation>.
2. tester si la valeur de cette variable ne dépasse pas la limite, si oui on sort du bloc
   si non on execute le bloc puis
3. {additionner / soustraire} suivant {+ / −} la valeur de <pas> à la variable
4. reprendre en (2).





Attention: le test est ~~executé~~ fait avant l'execution du bloc.
<pas> et <limite> sont des constantes ou des variables (pas d'expression).

exemple:

```
DØ FØR  I:= 2 STEP +3 UNTIL 15;
   :  (bloc executé 5 fois, pour les
   :   valeurs de I : 2, 5, 8, 11, 14.)
END;
```

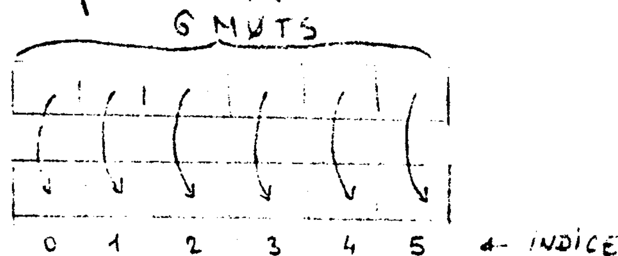
```
DØ FØR I:= 13 STEP -4 UNTIL 1;
   :  (bloc executé 4 fois, pour les
   :   valeurs de I : 13, 9, 5, 1.)
END;
```

EXEMPLE D'UTILISATIØN de L'INSTRUCTIØN DØ

Soit à transférer les 6 mots du tableau SOURCE dans les 6 mots du tableau DESTIN.

Ces 2 tableaux ont été déclarés auparavant :

```
ARRAY 6 WØRD SØURCE;
ARRAY 6 WØRD DESTIN;
```




```
I:=0;
DØ;
   DESTIN(I) := SØURCE(I);
   INCR I;
   EXIT ØN (I > 5)
END;
```

```
I:= 5;
DØ;
   DESTIN(I) := SØURCE (I);
   DECR I;
   EXIT ØN (I < 0)
END;
```

```
I:=0;
DØ WHILE (I <= 5);
   DESTIN(I):= SØURCE(I);
   INCR I
END;
```

```
I:= 5;
DØ WHILE (I >= 0);
   DESTIN (I) := SØURCE (I);
   DECR I
END;
```

```
I:= -6;
DØ INCR I TIMES;
   DESTIN (I+6) := SØURCE(I+6)
END;
```

```
I:= 6;
DØ DECR I TIMES;
   DESTIN (I-1) := SØURCE(I-1)
END;
```

```
DØ FØR I:=0 STEP +1 UNTIL 5;
   DESTIN(I):= SØURCE (I)
END;
```

```
DØ FØR I:=5 STEP -1 UNTIL 0;
   DESTIN(I):= SØURCE(I)
END;
```

LES ENTREES/SORTIES

Elles sont réalisées par des "procédures standards". Ces procédures ont déjà été compilés et se trouvent dans une bibliothèque.

Lecture d'une carte:

LICART (@ buffer)

L'appel de cette procédure fera, à l'exécution du programme, lire une carte perforée. L'argument de cette procédure est la constante adresse du tableau de bytes dans lequel on veut stocker les 80 caractères de la carte.

Impression d'un tableau de bytes.

IMPRIM (@ buffer, nombre)   Attention: IMPRIM n'a pas de E.

Le 1er argument de cette procédure est la constante adresse du tableau de bytes que l'on veut imprimer; le second est le nombre de bytes à imprimer.

exemple:

```
REF PRØCEDURE  LICART;   }
REF PRØCEDURE  IMPRIM;   }  déclarations
ARRAY 80 BYTE  BUFCART;  }
        :
        :
CALL LICART (@BUFCART);      }
CALL IMPRIM (@BUFCART, 80);  }  instructions
```

Il existe beaucoup d'autres procédures standards, en particulier toutes les conversions.

1 - LISTE DES MOTS RÉSERVÉS DU LANGAGE PL1600

	DETACH	IFIX	NSN	REQUEST	SKIP	U2
ACTIVATE	DO	IM	OF	RES	SLLD	U3
AND	DUMMY	IN	ON	RESET	SLRD	U4
ARM	EC	INCR	OR	RESSOURCE	SLLS	U5
	EL					
	ELSE				SLRS	
ARRAY	EM	INDIRECT	OUTPUT	RESTORE	SNAP	U6
ASSIGN	END	INPUT	OVERFLOW	REWIND	SO	UNTIL
ATTACH	ENTRY	IOCB	PC	RFL	SOFT	USING
BACKWARD	EOE	INSTRUCTION	POINTER	RK	STACK	WAIT
BEGIN	EOF	IS	PRIOR	RL	START	WHILE
BI		K7				
BIT	EU	KSTORE	PRIVATE	RSLE	STATUS	WORD
		LABEL			STOP	
BLOCK	EXECUTE	LL	PRMAX	RSLO	STEP	WORKING
BO		LO		RSNAP		
BYTE	EXIT	LOCAL	PROCÉDURE	RX	SWAP	WRITE
CA	EXT	LONG	PULL	RY	SYN	XOR
	FABS	LP		RW		
CALL	FILE	LPFILE	PUSH	SARD	TASK	ZE
					TEST	
CARRY	FIRST	MAIN	QUIT	SARS	THEN	
CASE	FLOAT	MASTER	RA	SAVE	TK	
CC	FLT	ME				
	FNEG					
COMMON	FOR	MESS	RAB	SCLD	TIMES	
CONSTANT	FORWARD	MESSMAX	RANK	SCRD	TO	
CONTINUE	FROM	MODE	RB	SCLS	TP	
CONTROL	GAP	MOVE	RC	SCRS	TR	
CR		MT				
CYCLE	GOTO	NEG	READ	SECTION	TRACE	
DATA	HARD	NEGATE	REF	SEGMENT	TS	
	HP					
DECR	HR	NORM	REGISTER	SET	TSYM	
	IB	NOSILENCE				
DEF	IF	NOT	RELEASE	SI	U1	
				SILENCE		

CODE TELETYPE (ASCII) PAR ORDRE NUMERIQUE

0	1	2	3
00 NULL	11 DC1	21 !	30 0 (zéro)
03 ETX	12 DC2 (perfo on)	22 "	33 3
05 ENQ	14 DC4 (perfo off)	24 $	35 5
06 ACK	17 ETB	27 '	36 6
09 HT	18 CAN	28 (	39 9
0A LF (line feed)	1B ESC	2B +	3A :
0C FF	1D GS	2D –	3C <
0F SI	1E RS	2E .	3F ?
4	**5**	**6**	**7**
41 A	50 P	60 `	71 q
42 B	53 S	63 c	72 r
44 D	55 U	65 e	74 t
47 G	56 V	66 f	77 w
48 H	59 Y	69 i	78 x
4B K	5A Z	6A j	7B
4D M	5C \ (diff. de /)	6C l	7D
4E N	5F ←	6F o	7E
8	**9**	**A**	**B**
81 SØH	90 DLE	A0 SP (espace)	B1 1
82 STX	93 DC3	A3 #	B2 2
84 EØT	95 NAK	A5 %	B4 4
87 BEL (sonnerie)	96 SYN	A6 &	B7 7
88 BS	99 EM	A9)	B8 8
8B VT	9A SUB	AA *	BB ;
8D RC (retour chariot)	9C FS	AC ,	BD =
8E SØ	9F US	AF /	BE >
C	**D**	**E**	**F**
C0 @	D1 Q	E1 a	F0 p
C3 C	D2 R	E2 b	F3 s
C5 E	D4 T	E4 d	F5 u
C6 F	D7 W	E7 g	F6 v
C9 I	D8 X	E8 h	F9 y
CA J	DB [	EB k	FA z
CC L	DD]	ED m	FC
CF Ø (lettre)	DE ↑	EE n	FF rub out

```
0  0   <<*********************************************************************
0  0   <<*********************************************************************
0  0   <<
0  0   <<                    E X E R C I C E   N O  1
0  0   <<
0  0   <<                    TRADUCTION DE CARTES.
0  0   <<
0  0   <<*********************************************************************
0  0   <<*********************************************************************
0  0
0  0
0  0   MAIN PROCEDURE EX01
1  1
1  1   .KSTORE SECTION POLLUX RES 50;
1  1
1  1   .LOCAL SECTION LOC
1  1
1  1        ARRAY 80 BYTE BUFCARTE=(" ");                 << BUFFER CARTE.
1  1        WORD I;                                        << POINTEUR SUR BUFCARTE.
1  1
1  1        REF PROCEDURE IMPRIM;                          << PROCEDURE D'IMPRESSION.
1  1        REF PROCEDURE LICART;                          << PROC DE LECTURE DE CARTE.
1  1
1  1   .USING KSTORE=POLLUX, LOCAL=LOC;
1  1
1  1        DO   WHILE (BUFCARTE(0) /= "/");               << TANT QUE 1ER CAR /= "/" ,
2  2             CALL LICART(@BUFCARTE);                   << LECTURE D'UNE CARTE.
2  2             CALL IMPRIM(@BUFCARTE,80);                << IMPRESSION TELLE QUE.
2  2             DO   FOR I:=0 STEP +1 UNTIL 79;           << POUR TOUS LES CARACTERES,
3  3                  IF   (BUFCARTE(I) = "&")             << SI "&" TRADUIRE
3  3                  THEN BUFCARTE(I):="E" ELSE           << EN "E"  SINON
5  4                  IF   (BUFCARTE(I) = '5F)             << SI BLANC SOULIGNE,
5  4                  THEN BUFCARTE(I):=" " END END        << TRADUIRE EN ESPACE.
6  3             END;                                      <<  DU ' DO FOR ... ' .
6  2             CALL IMPRIM(@BUFCARTE,80)                 << IMPRESSION CARTE TRADUITE.
6  2        END                                            <<  DU ' DO WHILE ... ' .
6  1
6  1   END.  <<  DE EX01.
```

E-1

*

```
 1  BUFCARTE                1    18  ARRAY BYTE , IN MAIN PROCEDURE EX01
               EX01                    26    27    28    30    31    32    33    35

 2  EX01                    0    12  MAIN PROCEDURE

 3  I                       1    19  WORD , IN MAIN PROCEDURE EX01
               EX01                    29    30    31    32    33

 4  IMPRIM                  1    21  REF PROCEDURE , IN MAIN PROCEDURE EX01
               EX01                    28    35

 5  LICART                  1    22  REF PROCEDURE , IN MAIN PROCEDURE EX01
               EX01                    27

 6  LOC                     1    16  LOCAL SECTION , IN MAIN PROCEDURE EX01
               EX01                    24

 7  POLLUX                  1    14  KSTORE SECTION , IN MAIN PROCEDURE EX01
               EX01                    24

FREEP=  2009
```

L'UNIVERS (QUE D'AUTRES APPELLENT LA BIBLIOTH&QUE) SE COMPOSE D'UN
L'UNIVERS (QUE D'AUTRES APPELLENT LA BIBLIOTHEQUE) SE COMPOSE D'UN
NOMBRE IND&FINI, ET PEUT-&TRE INFINI, DE GALERIES HEXAGONALES,
NOMBRE INDEFINI, ET PEUT-ETRE INFINI, DE GALERIES HEXAGONALES,
AVEC DE VASTES PUITS D'A&RATION AU MILIEU, BORD&ES PAR DES BALUSTRADES TR&S
AVEC DE VASTES PUITS D'AERATION AU MILIEU, BORDEES PAR DES BALUSTRADES TRES
BASSES. DE CHACUN DE CES HEXAGONES ON VOIT LES &TAGES INF&RIEURS ET
BASSES. DE CHACUN DE CES HEXAGONES ON VOIT LES ETAGES INFERIEURS ET
SUP&RIEURS, INTERMINABLEMENT. LA DISTRIBUTION DES GALERIES EST INVARIABLE.
SUPERIEURS, INTERMINABLEMENT. LA DISTRIBUTION DES GALERIES EST INVARIABLE.
VINGT-CINQ &TAG&RES, A RAISON DE CINQ PAR COT&, COUVRENT TOUS LES MURS
VINGT-CINQ ETAGERES, A RAISON DE CINQ PAR COTE, COUVRENT TOUS LES MURS
MOINS UN; LEUR HAUTEUR, QUI EST CELLE DES &TAGES EUX-M&MES, NE D&PASSE
MOINS UN; LEUR HAUTEUR, QUI EST CELLE DES ETAGES EUX-MEMES, NE DEPASSE
GU&RE CELLE D'UNE BIBLIOTH&QUE NORMALE. LE PLAN LIBRE DONNE SUR UN
GUERE CELLE D'UNE BIBLIOTHEQUE NORMALE. LE PLAN LIBRE DONNE SUR UN
COULOIR &TROIT, LEQUEL D&BOUCHE SUR UNE AUTRE GALERIE, IDENTIQUE A LA
COULOIR ETROIT, LEQUEL DEBOUCHE SUR UNE AUTRE GALERIE, IDENTIQUE A LA
PREMI&EE ET A TOUTES. A DROITE ET A GAUCHE DU COULOIR, IL Y A DEUX
PREMIEEE ET A TOUTES. A DROITE ET A GAUCHE DU COULOIR, IL Y A DEUX
CABINETS MINUSCULES. L'UN PERMET DE DORMIR DEBOUT_; L'AUTRE, DE
CABINETS MINUSCULES. L'UN PERMET DE DORMIR DEBOUT ; L'AUTRE, DE
SATISFAIRE LES BESOINS F&CAUX. C'EST ENTRE LES DEUX QUE PASSE L'ESCALIER
SATISFAIRE LES BESOINS FECAUX. C'EST ENTRE LES DEUX QUE PASSE L'ESCALIER
EN COLIMACON, QUI S'ABIME ET S'&L&VE A PERTE DE VUE. DANS LE COULOIR IL Y A
EN COLIMACON, QUI S'ABIME ET S'ELEVE A PERTE DE VUE. DANS LE COULOIR IL Y A
UN MIROIR, QUI DOUBLE FID&LEMENT LES APPARENCES. LES HOMMES EN TIRENT
UN MIROIR, QUI DOUBLE FIDELEMENT LES APPARENCES. LES HOMMES EN TIRENT
CONCLUSION QUE LA BIBLIOTH&QUE N'EST PAS INFINIE_; SI ELLE L'&TAIT
CONCLUSION QUE LA BIBLIOTHEQUE N'EST PAS INFINIE ; SI ELLE L'ETAIT
R&ELLEMENT, A QUOI BON CETTE DUPLICATION ILLUSOIRE_?
REELLEMENT, A QUOI BON CETTE DUPLICATION ILLUSOIRE ?
MOI JE PR&F&RE R&VER QUE CES SURFACES POLIES SONT LA POUR FIGURER
MOI JE PREFERE REVER QUE CES SURFACES POLIES SONT LA POUR FIGURER
L'INFINI ET POUR LE PROMETTRE_...
L'INFINI ET POUR LE PROMETTRE ...

E-3

```
0  0
0  0  <<*******************************************************************************
0  0  <<*******************************************************************************
0  0  <<
0  0  <<                   E X E R C I C E   N O  2
0  0  <<
0  0  <<              SEPARATION DES MOTS D'UN TEXTE LU SUR CARTE.
0  0  <<
0  0  <<*******************************************************************************
0  0  <<*******************************************************************************
0  0
0  0
0  0  MAIN PROCEDURE EX02
1  1
1  1  .KSTORE SECTION POLLUX RES 50;
1  1
1  1  .LOCAL SECTION LOC
1  1
1  1       ARRAY 80 BYTE BUFCARTE;                        << BUFER CARTE.
1  1       WORD POBUF=(79);                               << POINTEUR COURANT SUR BUFCAR
1  1       ARRAY 80 BYTE MOT=(" ");                       << CONTIENT LE MOT LU.
1  1       WORD POMOT;                                    << POINTEUR SUR MOT.
1  1       BYTE CARACTERE;                                << CONTIENT LE CARACTERE LU.
1  1
1  1       REF PROCEDURE IMPRIM;                          << PROCEDURE D'IMPRESSION.
1  1       REF PROCEDURE LICART;                          << PROC DE LECTURE DE CARTE.
1  1
1  1
1  1
1  1       <<------------------------------------------------------------------------
1  1       <<-----      METS LE CARACTERE SUIVANT DANS "CARACTERE" .
1  1       <<------------------------------------------------------------------------
1  1
1  1       PROCEDURE CARSUIV
2  2       .USING LOCAL IS LOC;
2  2            INCR POBUF;                               << INCREM POINTEUR BUFFER CARTE
2  2            IF   (POBUF > 79)                         << FIN DE CARTE ?
2  2            THEN CALL LICART(@BUFCARTE);              << OUI, ON RELIT UNE CARTE E
3  3                 POBUF:=0 END;                        << ON POSITIONNE POBUF.
3  2            CARACTERE:=BUFCARTE(POBUF);               << CHARGEMENT "CARACTERE" .
3  2       END;   <<  DE CARSUIV .
```


E-4

```
3   1
3   1
3   1   <<******************************************************************************
3   1   <<**********          P R O C E D U R E   P R I N C I P A L E          **********
3   1   <<******************************************************************************
3   1
3   1   .USING KSTORE=POLLUX,LOCAL=LOC;
3   1
3   1        DO   WHILE (MOT(O) /= "/");                       << TANT QUE LE 1ER CAR /= "/" ,
4   2             BEGIN CALL CARSUIV;                          << APPEL CARACTERE SUIVANT.
5   3                   CYCLE ON (CARACTERE = " ")             << ON SAUTE TOUS LES ESPACES.
5   3             END;                                         <<  DU ' BEGIN ... ' .
5   2             POMOT:=O;                                    << POSITIONNEMENT POMOT.
5   2             BEGIN MOT(POMOT):=CARACTERE;                 << CHARGEMENT DU CARACTERE.
6   3                   INCR POMOT;                            << PREPARATION CARACTERE SUIVAN
6   3                   CALL CARSUIV;                          << APPEL CARACTERE SUIVANT.
6   3                   CYCLE ON (CARACTERE /= " ")            << BOUCLE SI DIFFERENT DE ESPAC
6   3             END;                                         <<  DU ' BEGIN ... ' .
6   2             CALL IMPRIM(@MOT,POMOT)                      << IMPRESSION DU MOT.
6   2        END                                               <<  DU ' DO WHILE ... ' .
6   1   END.  <<  DE EXO2 .
```

L'UNIVERS
(QUE
D'AUTRES
APPELLENT
LA
BIBLIOTH&QUE)
SE
COMPOSE
D'UN
NOMBRE
IND&FINI,
ET
PEUT-&TRE
INFINI,
DE
GALERIES
HEXAGONALES,
AVEC
DE
VASTES
PUITS
D'A&RATION
AU
MILIEU,
BORD&ES
PAR
DES
BALUSTRADES
TR&S
BASSES.
DE
CHACUN
DE
CES
HEXAGONES
ON
VOIT
LES
&TAGES
INF&RIEURS
ET
SUP&RIEURS,
INTERMINABLEMENT.
LA
DISTRIBUTION
DES
GALERIES

```
<<********************************************************************
<<********************************************************************
<<
<<              E X E R C I C E    N O   3
<<
<<         EDITION D'UN TEXTE LU SUR CARTE
<<         DANS UN PAVE, SANS JUSTIFICATION.
<<
<<********************************************************************
<<********************************************************************


MAIN PROCEDURE EX03

.KSTORE SECTION POLLUX RES 50;                        << ZONE SYSTEME.

.LOCAL SECTION LOC                                    << DECLARATION + RESERVATION

     ARRAY 80 BYTE BUFCARTE;                          << BUFFER CARTE.
     WORD POBUF=(79);                                 << POINTEUR COURANT SUR BUFCART

     BYTE CARACTERE;                                  << CONTIENT LE CARACTERE LU.

     ARRAY 80 BYTE MOT=(" ");                         << CONTIENT LE DERNIER MOT LU.
     WORD POMOT;                                      << POINTEUR COURANT SUR "MOT" .

     ARRAY 120 BYTE LIGNE;                            << CONTIENT LA LIGNE EN CONSTRU
     WORD POLIG=(0);                                  << POINTEUR COURANT SUR "LIGNE"

     WORD LONGJUST=(60);                              << LONGEUR DE LA JUSTIFICATION.

     WORD I;                                          << AUXILLIAIRE.

     REF PROCEDURE LICART;                            << PROCEDURE DE LECTURE DE CART
     REF PROCEDURE IMPRIM;                            << PROCEDURE D'IMPRESSION.

%PAGE
```

E-8

```
1  1       <<-----------------------------------------------------------------------------
1  1       <<-----      METS LE CARACTERE SUIVANT DANS  "CARACTERE" .
1  1       <<-----------------------------------------------------------------------------
1  1
1  1       PROCEDURE CARSUIV
2  2       .USING LOCAL IS LOC;
2  2             INCR POBUF;                               << INCREM POINTEUR BUFFER.
2  2             IF    (POBUF > 79)                        << SI FIN DE CARTE.
2  2             THEN CALL LICART(@BUFCARTE);              << ALORS ON RELIT UNE CARTE ET
3  3                  POBUF:=0 END;                        << ON POSITIONNE POBUF.
3  2             CARACTERE:=BUFCARTE(POBUF)                << CHARGEMENT "CARACTERE" .
3  2       END;  <<   DE CARSUIV.
3  1
3  1
3  1       <<-----------------------------------------------------------------------------
3  1       <<-----      METS LE MOT SUIVANT DANS  "MOT" .
3  1       <<-----------------------------------------------------------------------------
3  1
3  1       PROCEDURE MOTSUIV
4  2       .USING LOCAL IS LOC;
4  2             BEGIN CALL CARSUIV;                       << ON SAUTE TOUS LES
5  3                   CYCLE ON (CARACTERE = " ")          << ESPACES.
5  3             END;                                      <<  DU ' BEGIN ... ' .
5  2             POMOT:=0;                                 << POSITIONNEMENT POMOT.
5  2             DO;  MOT(POMOT):=CARACTERE;               << CHARGEMENT MOT.
6  3                  EXIT ON (CARACTERE = " ");           << ON SORT SI ESPACE.
6  3                  CALL CARSUIV;                        << APPEL CARACTERE SUIVANT.
6  3                  INCR POMOT                           << INCREM POINTEUR MOT.
6  3             END                                       <<  DU ' DO ... ' .
6  2       END;  <<   DE MOTSUIV.
6  1
6  1    %PAGE
```

6-3

```
 6  1   <<----------------------------------------------------------------------------------
 6  1   <<-----       PREPARE LA LIGNE ET L'IMPRIME .
 6  1   <<----------------------------------------------------------------------------------
 6  1
 6  1     PROCEDURE ENVOYER
 7  2     .USING LOCAL IS LOC;
 7  2          DO    FOR I:=0 STEP +1 UNTIL POLIG;       << POUR TOUS LES CARACTERES ,
 8  3                IF    (LIGNE(I)  =  "&")            << SI "&" TRADUIRE
 8  3                THEN LIGNE(I):="E" ELSE             << EN "E" .
10  4                IF    (LIGNE(I)  =  '5F)            << SI BLANS SOULIGNE,
10  4                THEN LIGNE(I):=" " END END          << TRADUIRE EN ESPACE.
11  3          END;                                      <<  DU ' DO FOR ... ' .
11  2          CALL  IMPRIM(@LIGNE,POLIG)                << IMPRIMER LA LIGNE.
11  2     END;  <<  DE ENVOYER.
11  1
11  1
11  1   <<****************************************************************************
11  1   <<**********        P R O C E D U R E   P R I N C I P A L E        **********
11  1   <<****************************************************************************
11  1
11  1   .USING KSTORE=POLLUX, LOCAL=LOC;
11  1
11  1     DO    WHILE (MOT(0)  /=  "/");                 << FAIRE TANT QUE 1CAR MOT/= /
12  2          CALL  MOTSUIV;                            << APPEL MOT SUIVANT.
12  2          IF    (LONGJUST < POLIG+POMOT)            << ON SORT DE LA LIGNE,
12  2          THEN CALL ENVOYER;                        << OUI, ON L'IMPRIME ET
13  3                POLIG:=0 END;                       << ON POSIITONNE POLIG.
13  2          DO    FOR I:=0 STEP +1 UNTIL POMOT;       << CHARGEMENT DU MOT DANS LA LI
14  3                LIGNE(POLIG):=MOT(I);               << LIGNE.
14  3                INCR POLIG                          << INCREM POINTEUR LIGNE.
14  3          END                                       <<  DU ' DO FOR ... ' .
14  2     END;                                           <<  DU ' DO WHILE ... ' .
14  1     CALL ENVOYER                                   << IMPRESSION DERNIERE LIGNE.
14  1   END.  <<  DE EX03 .
```

E-10

```
  1  BUFCARTE                1     20  ARRAY BYTE , IN MAIN PROCEDURE EXO3
                CARSUIV                  47    49

  2  CARACTERE               1     23  BYTE , IN MAIN PROCEDURE EXO3
                CARSUIV                  49
                MOTSUIV                  60    63    64

  3  CARSUIV                 1     43  PROCEDURE , IN MAIN PROCEDURE EXO3
                MOTSUIV                  59    65

  4  ENVOYER                 1     75  PROCEDURE , IN MAIN PROCEDURE EXO3
                EXO3                     96   103

  5  EXO3                    0     14  MAIN PROCEDURE

  6  I                       1     33  WORD , IN MAIN PROCEDURE EXO3
                ENVOYER                  77    78    79    80    81
                EXO3                     98    99

  7  IMPRIM                  1     36  REF PROCEDURE , IN MAIN PROCEDURE EXO3
                ENVOYER                  83

  8  LICART                  1     35  REF PROCEDURE , IN MAIN PROCEDURE EXO3
                CARSUIV                  47

  9  LIGNE                   1     28  ARRAY BYTE , IN MAIN PROCEDURE EXO3
                ENVOYER                  78    79    80    81    83
                EXO3                     99

 10  LOC                     1     18  LOCAL SECTION , IN MAIN PROCEDURE EXO3
                CARSUIV                  44
                MOTSUIV                  58
                ENVOYER                  76
                EXO3                     91

 11  LONGJUST                1     31  WORD , IN MAIN PROCEDURE EXO3
                EXO3                     95

 12  MOT                     1     25  ARRAY BYTE , IN MAIN PROCEDURE EXO3
                MOTSUIV                  63
```

E.44

```
                 EX03                       93    99

  13  MOTSUIV                  1     57  PROCEDURE , IN MAIN PROCEDURE EX03
                 EX03                       94

  14  POBUF                    1     21  WORD , IN MAIN PROCEDURE EX03
                 CARSUIV                    45    46    48    49

  15  POLIG                    1     29  WORD , IN MAIN PROCEDURE EX03
                 ENVOYER                    77    83
                 EX03                       95    97    99   100

  16  POLLUX                   1     16  KSTORE SECTION , IN MAIN PROCEDURE EX03
                 EX03                       91

  17  POMOT                    1     26  WORD , IN MAIN PROCEDURE EX03
                 MOTSUIV                    62    63    66
                 EX03                       95    98

FREEP=  2211
```

E-12

L'UNIVERS (QUE D'AUTRES APPELLENT LA BIBLIOTHEQUE) SE COMPOSE D'UN NOMBRE INDEFINI, ET PEUT-ETRE INFINI, DE GALERIES HEXAGONALES, AVEC DE VASTES PUITS D'AERATION AU MILIEU, BORDEES PAR DES BALUSTRADES TRES BASSES. DE CHACUN DE CES HEXAGONES ON VOIT LES ETAGES INFERIEURS ET SUPERIEURS, INTERMINABLEMENT. LA DISTRIBUTION DES GALERIES EST INVARIABLE. VINGT-CINQ ETAGERES, A RAISON DE CINQ PAR COTE, COUVRENT TOUS LES MURS MOINS UN; LEUR HAUTEUR, QUI EST CELLE DES ETAGES EUX-MEMES, NE DEPASSE GUERE CELLE D'UNE BIBLIOTHEQUE NORMALE. LE PLAN LIBRE DONNE SUR UN COULOIR ETROIT, LEQUEL DEBOUCHE SUR UNE AUTRE GALERIE, IDENTIQUE A LA PREMIEEE ET A TOUTES. A DROITE ET A GAUCHE DU COULOIR, IL Y A DEUX CABINETS MINUSCULES. L'UN PERMET DE DORMIR DEBOUT ; L'AUTRE, DE SATISFAIRE LES BESOINS FECAUX. C'EST ENTRE LES DEUX QUE PASSE L'ESCALIER EN COLIMACON, QUI S'ABIME ET S'ELEVE A PERTE DE VUE. DANS LE COULOIR IL Y A UN MIROIR, QUI DOUBLE FIDELEMENT LES APPARENCES. LES HOMMES EN TIRENT CONCLUSION QUE LA BIBLIOTHEQUE N'EST PAS INFINIE ; SI ELLE L'ETAIT REELLEMENT, A QUOI BON CETTE DUPLICATION ILLUSOIRE ? MOI JE PREFERE REVER QUE CES SURFACES POLIES SONT LA POUR FIGURER L'INFINI ET POUR LE PROMETTRE ... COMME TOUS LES HOMMES DE LA BIBLIOTHEQUE, J'AI VOYAGE DANS MA JEUNESSE ; J'AI EFFECTUE DES PELERINAGES, A LA RECHERCHE D'UN LIVRE ET PEUT-ETRE DU CATALOGUE DES CATALOGUES; MAINTENANT QUE MES YEUX NE PEUVENT PRESQUE PLUS DECHIFFRER CE QUE J'ECRIS, JE ME PREPARE A MOURIR A QUELQUES COURTES LIEUX DE L'HEXAGONE OU JE NAQUIS. MORT, IL NE MANQUERA PAS DE MAINS PIEUSES POUR ME JETER PAR-DESSUS LA BALUSTRADE; MON TOMBEAU SERA L'AIR INSONDABLE; MON CORPS S'ENFONCERA LONGUEMENT, ET IL SE CORROMPRA ET IL SE DISSOUDRA DANS LE VENT ENGENDRE PAR LA CHUTE, QUI EST INFINIE. CAR J'AFFIRME QUE LA BIBLIOTHEQUE EST INTERMINABLE. LES IDEALISTES ARGUMENTENT QUE LES SALLESHEXAGONALES SONT UNE FORME NECESSAIRE DE L'ESPACE ABSOLU, OU DU MOINS DE NOTRE INTUITION DE L'ESPACE. ILS ESTIMENT QU'UNE SALLE TRIANGULAIRE OU PENTAGONALE SERAIT INCONCEVABLE. (LES MYSTIQUES PRETENDENT QUE L'EXTASE LEUR REVELE UNE CHAMBRE CIRCULAIRE AVEC UN GRAND LIVRE CIRCULAIRE A DOS CONCAVE CONTINU, QUI FAIT TOUT LE TOUR DES MURS; MAIS LEUR TEMOIGNAGE EST SUSPECT, ET LEURS PAROLES OBSCURES: CE LIVRECYCLIQUE, C'EST DIEU.) QU'IL ME SUFFISE, POUR LE MOMENT, DE REPE TER LA SENTENCE CLASSIQUE: LA BIBLIOTHEQUE EST UNE SPHERE DONT LE CENTRE VERITABLE EST UN HEXAGONE QUELCONQUE, ET LA CIRCONFERENCE INACCESSIBLE. /

```
0 0
0 0  <<****************************************************************
0 0  <<****************************************************************
0 0  <<
0 0  <<                  E X E R C I C E    N O   4
0 0  <<
0 0  <<              EDITION D'UN TEXTE LU SUR CARTE
0 0  <<              DANS UN PAVE JUSTIFIE.
0 0  <<
0 0  <<****************************************************************
0 0  <<****************************************************************
0 0
0 0
0 0
0 0  MAIN PROCEDURE EX04
1 1
1 1  .KSTORE SECTION POLLUX RES 50;
1 1
1 1  .LOCAL SECTION LOC
1 1
1 1       ARRAY 80 BYTE BUFCARTE;            << BUFFER CARTE.
1 1       WORD POBUF=(79);                   << POINTEUR COURANT SUR BUFCART
1 1
1 1       BYTE CARACTERE;                    << CONTIENT LE CARACTERE LU.
1 1
1 1       ARRAY 80 BYTE MOT;                 << CONTIENT LE MOT LU.
1 1       WORD POMOT;                        << POINTEUR COURANT SUR MOT.
1 1
1 1       ARRAY 120 BYTE LIGNE;              << CONTIENT LA LIGNE EN CONSTRU
1 1       WORD POLIG=(0);                    << POINTEUR COURANT SUR LIGNE.
1 1
1 1       WORD LONGJUST=(40);                << LONGEUR DE LA JUSTIFICATION.
1 1       WORD NBMOT=(0);                    << NB DE MOTS DANS LA LIGNE.
1 1       WORD I, J, K;                      << AUXILLIAIRES.
1 1
1 1       REF PROCEDURE LICART;              << PROC DE LECTURE DE CA1IE.
1 1       REF PROCEDURE IMPRIM;              << PROCEDURE D'IMPRESSION.
1 1
1 1  %PAGE
```

E.14

```
1  1
1  1  <<-----------------------------------------------------------------
1  1  <<-----       METS LE CARACTERE SUIVANT DANS "CARACTERE" .
1  1  <<-----------------------------------------------------------------
1  1
1  1  PROCEDURE CARSUIV
2  2  .USING LOCAL IS LOC;
2  2       INCR POBUF;                              << INCREM POINTEUR BUFFER CARTE
2  2       IF   (POBUF > 79)                        << FIN DE CARTE ?
2  2       THEN CALL LICART(@BUFCARTE);             << OUI, ON RELIT UNE CARTE ET
3  3            POBUF:=0 END;                       << ON POSITIONNE POBUF.
3  2       CARACTERE:=BUFCARTE(POBUF)               << CHARGEMENT "CARACTERE" .
3  2  END;   <<  DE CARSUIV .
3  1
3  1
3  1
3  1  <<-----------------------------------------------------------------
3  1  <<-----       METS LE MOT SUIVANT DANS "MOT" .
3  1  <<-----------------------------------------------------------------
3  1
3  1  PROCEDURE MOTSUIV
4  2  .USING LOCAL IS LOC;
4  2       BEGIN CALL CARSUIV;                      << ON SAUTE TOUS LES
5  3             CYCLE ON (CARACTERE = " ")         << ESPACES.
5  3       END;                                     <<   DU ' BEGIN ... ' .
5  2       POMOT:=0;                                << POSITIONNEMENT POMOT.
5  2       DO;  MOT(POMOT):=CARACTERE;              << CHARGEMENT MOT.
6  3            EXIT ON (CARACTERE = " ");          << ON SORT SI ESPACE.
6  3            CALL CARSUIV;                       << APPEL CARACTERE SUIVANT.
6  3            INCR POMOT                          << INCREM POINTEUR MOT;
6  3       END                                      <<   DU ' DO ... ' .
6  2  END;   <<  DE MOTSUIV .
6  1
6  1  %PAGE
```

```
 6   1
 6   1   <<-----------------------------------------------------------------
 6   1   <<-----     JUSTIFICATION DE LA LIGNE.
 6   1   <<-----------------------------------------------------------------
 6   1
 6   1   PROCEDURE JUSTIF
 7   2
 7   2   .LOCAL SECTION CONTINUE LOC
 7   2        WORD VALESPM;                               << VALEUR DE L'ESPACE-MOT.
 7   2        WORD RESTE;                                 << RESTE DE LA DIVISION ENTIERE
 7   2
 7   2   .USING LOCAL IS LOC;
 7   2   LABEL FJUST;
 7   2
 7   2        DECR NBMOT;                                 << NB D'ESPACES-MOT.
 7   2        GOTO FJUST ON (NBMOT <= 0);                 << ON PEUT PAS JUSTIFIER.
 7   2        VALESPM:=LONGJUST-POLIG+1/NBMOT+1;          << CALCUL VALEUR ESPACE-MOT.
 7   2        RESTE:=RB;                                  << RECUPERATION DU RESTE / .
 7   2        J:=LONGJUST-1;                              << PREP BOUCLE.
 7   2        DO   FOR I:=POLIG-2 STEP -1 UNTIL 1;        << BOUCLE SUR LIGNE.
 8   3             IF   (LIGNE(I) = " ")                  << C'EST UN ESPACE-MOT.
 8   3             THEN IF   (RESTE = 0)                  << YA PLUS DE RESTE ?
 9   4                  THEN DECR VALESPM END;            << OUAIP, ON DECR VALESPM.
10   4                  DECR RESTE;                       << ON DECREM RESTE.
10   4                  DO   FOR K:=0 STEP +1 UNTIL VALESPM;   << CHARGE ESPACES.
11   5                       LIGNE(J):=" "; DECR J             <<
11   5                  END                               <<   DU ' DO FOR ... ' .
11   4             ELSE LIGNE(J):=LIGNE(I);               << ON CHARGE LE CARACTERE.
12   4                  DECR J
12   4             END                                    <<   DU ' IF ... ' .
12   3        END;                                        <<   DU ' DO FOR ... ' .
12   2        POLIG:=LONGJUST;                            << PREP 'ENVOYER' .
12   2   FJUST:
12   2        CALL ENVOYER                                << ON ENVOIE LA LIGNE.
12   2     END;   <<  DE JUSTIFICATION.
12   1
12   1  %PAGE
```

```
12  1      <<-----------------------------------------------------------------------
12  1      <<---    TRADUIT LA LIGNE ET L'IMPRIME.
12  1      <<-----------------------------------------------------------------------
12  1
12  1      PROCEDURE ENVOYER
13  2      .USING LOCAL IS LOC;
13  2            DO    FOR I:=0 STEP +1 UNTIL POLIG;          << POUR TOUTE LA LIGNE.
14  3                  IF    (LIGNE(I)  =  "&")               <<   SI "&" TRADUIRE
14  3                  THEN LIGNE(I)  :=  "E"                 <<   EN "E",
15  4                  ELSE IF    (LIGNE(I)  =  '5F)          <<   SI BLANC SOULIGNE,
16  4                        THEN LIGNE(I):=" "  END END      <<   TRADUIRE EN ESPACE.
17  3            END;                                         <<   DU ' DO FOR ... ' .
17  2            CALL IMPRIM(@LIGNE,POLIG);                   << IMPRIME LA LIGNE.
17  2            POLIG, NBMOT :=0                             << PREPARATION NOUVELLE LIGNE.
17  2      END;    <<  DE ENVOYER.
17  1
17  1
17  1   <<**********************************************************************
17  1   <<***********          P R O C E D U R E   P R I N C I P A L E    ***********
17  1   <<**********************************************************************
17  1
17  1   .USING KSTORE=POLLUX,LOCAL=LOC;
17  1
17  1        DO;   CALL MOTSUIV;                               << APPEL MOT SUIVANT.
18  2              IF    (LONGJUST < POLIG+POMOT)              << CA RENTRE PAS ?
18  2              THEN CALL JUSTIF END;                       << APPEL JUSTIFICATION.
19  2              INCR NBMOT;
19  2              DO    FOR I:=0 STEP +1 UNTIL POMOT;         << CHARGEMENT DU MOT
20  3                    LIGNE(POLIG):=MOT(I);                 << DANS LA LIGNE.
20  3                    INCR POLIG
20  3              END;
20  2              EXIT ON (MOT(0)  =  "/");                   <<   DU ' DO FOR ... ' .
20  2              IF    (POLIG = LONGJUST+1)                  << FIN DU TEXTE.
20  2              THEN CALL ENVOYER END                       << CA RENTRE JUSTE.
21  2        END
21  1   END.   <<  DE EX04.                                   <<   DU ' DO ... ' .
```

```
  1  BUFCARTE                  1    21  ARRAY BYTE , IN MAIN PROCEDURE EX04
                CARSUIV                   49    51

  2  CARACTERE                 1    24  BYTE , IN MAIN PROCEDURE EX04
                CARSUIV                   51
                MOTSUIV                   63    66    67

  3  CARSUIV                   1    45  PROCEDURE , IN MAIN PROCEDURE EX04
                MOTSUIV                   62    68

  4  ENVOYER                   1   115  PROCEDURE , IN MAIN PROCEDURE EX04
                JUSTIF                   107
                EX04                     144

  5  EX04                      0    15  MAIN PROCEDURE

  6  FJUST                     2    86  LABEL , IN PROCEDURE JUSTIF
                JUSTIF                    89   106

  7  I                         1    34  WORD , IN MAIN PROCEDURE EX04
                JUSTIF                    93    94   101
                ENVOYER                  117   118   119   120   121
                EX04                     138   139

  8  IMPRIM                    1    37  REF PROCEDURE , IN MAIN PROCEDURE EX04
                ENVOYER                  123

  9  J                         1    34  WORD , IN MAIN PROCEDURE EX04
                JUSTIF                    92    99    99   101   102

 10  JUSTIF                    1    79  PROCEDURE , IN MAIN PROCEDURE EX04
                EX04                     136

 11  K                         1    34  WORD , IN MAIN PROCEDURE EX04
                JUSTIF                    98

 12  LICART                    1    36  REF PROCEDURE , IN MAIN PROCEDURE EX04
                CARSUIV                   49
```

E-12

```
13  LIGNE                  1   29  ARRAY BYTE , IN MAIN PROCEDURE EX04
               JUSTIF               94    99   101   101
               ENVOYER             118   119   120   121   123
               EX04                139

14  LOC                    1   19  LOCAL SECTION , IN MAIN PROCEDURE EX04
               CARSUIV              46
               MOTSUIV              61
               JUSTIF               81    85
               ENVOYER             116
               EX04                132

15  LONGJUST               1   32  WORD , IN MAIN PROCEDURE EX04
               JUSTIF               90    92   105
               EX04                135   143

16  MOT                    1   26  ARRAY BYTE , IN MAIN PROCEDURE EX04
               MOTSUIV              66
               EX04                139   142

17  MOTSUIV                1   60  PROCEDURE , IN MAIN PROCEDURE EX04
               EX04                134

18  NBMOT                  1   33  WORD , IN MAIN PROCEDURE EX04
               JUSTIF               88    89    90
               ENVOYER             124
               EX04                137

19  POBUF                  1   22  WORD , IN MAIN PROCEDURE EX04
               CARSUIV              47    48    50    51

20  POLIG                  1   30  WORD , IN MAIN PROCEDURE EX04
               JUSTIF               90    93   105
               ENVOYER             117   123   124
               EX04                135   139   140   143

21  POLLUX                 1   17  KSTORE SECTION , IN MAIN PROCEDURE EX04
               EX04                132

22  POMOT                  1   27  WORD , IN MAIN PROCEDURE EX04
               MOTSUIV              65    66    69
```

E-19

```
                EX04                135    138

  23  RESTE                2    83  WORD , IN PROCEDURE JUSTIF
                JUSTIF                  91     95     97

  24  VALESPM              2    82  WORD , IN PROCEDURE JUSTIF
                JUSTIF                  90     96     98

FREEP=  2390
```

L'UNIVERS (QUE D'AUTRES APPELLENT LA BIBLIOTHEQUE) SE COMPOSE D'UN NOMBRE INDEFINI, ET PEUT-ETRE INFINI, DE GALERIES HEXAGONALES, AVEC DE VASTES PUITS D'AERATION AU MILIEU, BORDEES PAR DES BALUSTRADES TRES BASSES. DE CHACUN DE CES HEXAGONES ON VOIT LES ETAGES INFERIEURS ET SUPERIEURS, INTERMINABLEMENT. LA DISTRIBUTION DES GALERIES EST INVARIABLE. VINGT-CINQ ETAGERES, A RAISON DE CINQ PAR COTE, COUVRENT TOUS LES MURS MOINS UN; LEUR HAUTEUR, QUI EST CELLE DES ETAGES EUX-MEMES, NE DEPASSE GUERE CELLE D'UNE BIBLIOTHEQUE NORMALE. LE PLAN LIBRE DONNE SUR UN COULOIR ETROIT, LEQUEL DEBOUCHE SUR UNE AUTRE GALERIE, IDENTIQUE A LA PREMIEEE ET A TOUTES. A DROITE ET A GAUCHE DU COULOIR, IL Y A DEUX CABINETS MINUSCULES. L'UN PERMET DE DORMIR DEBOUT ; L'AUTRE, DE SATISFAIRE LES BESOINS FECAUX. C'EST ENTRE LES DEUX QUE PASSE L'ESCALIER EN COLIMACON, QUI S'ABIME ET S'ELEVE A PERTE DE VUE. DANS LE COULOIR IL Y A UN MIROIR, QUI DOUBLE FIDELEMENT LES APPARENCES. LES HOMMES EN TIRENT CONCLUSION QUE LA BIBLIOTHEQUE N'EST PAS INFINIE ; SI ELLE L'ETAIT REELLEMENT, A QUOI BON CETTE DUPLICATION ILLUSOIRE ? MOI JE PREFERE REVER QUE CES SURFACES POLIES SONT LA POUR FIGURER L'INFINI ET POUR LE PROMETTRE ... COMME TOUS LES HOMMES DE LA BIBLIOTHEQUE, J'AI VOYAGE DANS MA JEUNESSE ; J'AI EFFECTUE DES PELERINAGES, A LA RECHERCHE D'UN LIVRE ET PEUT-ETRE DU CATALOGUE DES CATALOGUES; MAINTENANT QUE MES YEUX NE PEUVENT PRESQUE PLUS DECHIFFRER CE QUE J'ECRIS, JE ME PREPARE A MOURIR A QUELQUES COURTES LIEUX DE L'HEXAGONE OU JE NAQUIS. MORT, IL NE MANQUERA PAS DE MAINS PIEUSES POUR ME JETER PAR-DESSUS LA BALUSTRADE; MON TOMBEAU SERA L'AIR INSONDABLE; MON CORPS S'ENFONCERA LONGUEMENT, ET IL SE CORROMPRA ET IL SE DISSOUDRA DANS LE VENT ENGENDRE PAR LA CHUTE, QUI EST INFINIE. CAR J'AFFIRME QUE LA BIBLIOTHEQUE EST INTERMINABLE